Migrating from Per-Job Analysis to Per-Resource Analysis for Tighter Bounds of End-to-End Response Times

Man-Ki Yoon, *Student Member, IEEE,* Chang-Gun Lee, *Member, IEEE,* and Junghee Han

Abstract—As the software complexity drastically increases for multi-resource real-time systems, industries have great needs for analytically validating real-time behaviors of their complex software systems. Possible candidates for such analytic validations are the end-to-end response time analysis techniques that can analytically find the worst case response times of real-time transactions over multiple resources. The existing techniques, however, exhibit severe overestimation when real-time transactions visit the same resource multiple times, which we call a *multiple visit problem*. To address the problem, this paper proposes a novel analysis that completely changes its analysis viewpoint from classical *per-job basis*—aggregation of per-job response times—to *per-resource basis*—aggregation of per-resource total delays. Our experiments show that the proposed analysis can find significantly tighter bounds of end-to-end response times compared with the existing per-job based analysis.

Index Terms—Per-resource analysis, end-to-end response time analysis, controller area network, real-time and embedded systems.

1 Introduction

Recently, many cyber-physical systems such as automobiles and aircrafts are increasingly employing electronic parts because of their functional diversity and low-cost. Accordingly, the complexity of software that works with those electronic parts is also increasing. Thus, industries have great needs for analytically validating real-time behaviors of such complex software systems.

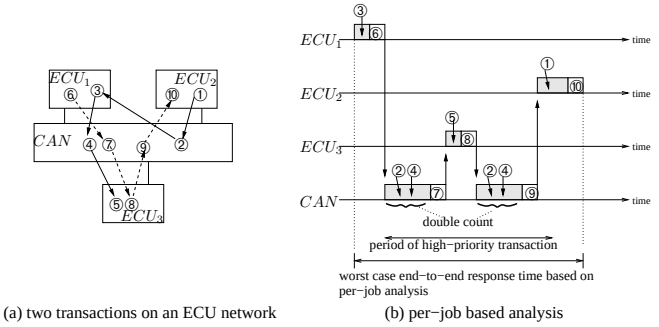
For that purpose, we may use the holistic analysis technique proposed by Tindell [1] and its extensions [2], [3], [4], [5], [6], [7] that can analytically compute the end-to-end response times of transactions that consist of a sequence of tasks running over chains of multiple resources such as electronic sensors / actuators, networks, and microprocessors. However, we identify that the existing techniques give very loose bounds on the end-to-end response time when a transaction visits the same resource multiple times, which we call a *multiple visit problem*.

- *M.-K. Yoon is with the Department of Computer Science, Thomas M. Siebel Center for Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801-2302. E-mail: mkyoon@uiuc.edu*
- *C.-G. Lee is with the School of Computer Science and Engineering, Bldg. 301 Room 454-2, Seoul National University, 599 Gwanangno, Gwanak-gu, Seoul, 151-742, Korea. E-mail: cglee@snu.ac.kr*
- *J. Han is with the Department of Telecommunication and Computer Engineering, Korea Aerospace University, 100 Hanggongdae-gil Hwajeon-dong, Deokyang-Gu, Goyang-City, Gyeonggi-do 412-791, Korea. E-mail: junghee@kau.ac.kr*

Manuscript received 16 Jan. 2009; revised 18 Sep. 2009; accepted 27 Sep. 2009; published online X XXX. 20XX.
Recommended for acceptance by
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number Digital Object Identifier no.

As a simple example to motivate the multiple visit problem, let us consider Fig. 1(a), which shows three ECUs (Electronic Control Units) connected through a CAN (Controller Area Network) bus. On top of these resources, a high priority periodic transaction consists of five tasks (①,②,③,④,⑤) that visit ECU_2, CAN, ECU_1, CAN, and ECU_3, respectively. In addition, a low priority periodic transaction consists of five tasks (⑥,⑦,⑧,⑨,⑩) that visit ECU_1, CAN, ECU_3, CAN, and ECU_2, respectively. For this system, existing techniques compute the response time of each job assuming the worst case scenario of high priority arrivals and then add up all the per-job response times to compute the end-to-end response time as shown in Fig. 1(b). Such per-job based analysis, however, severely double-counts the high priority arrivals when a low priority transaction visits the same resource multiple times. In the example, the low priority transaction visits CAN twice, i.e., jobs ⑦ and ⑨. For each of these two visits, the per-job based analysis assumes the worst case delay by the high-priority jobs on CAN, i.e., ② and ④. Thus, execution times of ② and ④ are doubly-counted in the end-to-end response time of the low-priority transaction as shown in Fig. 1(b). However, considering the period of the high priority transaction, we can note that at most a single instance of the high priority transaction can delay the given instance of the low priority transaction at CAN.

This overestimation becomes more serious as the multiple visit count becomes larger due to complex long transactions. Due to this reason, the traditional per-job based end-to-end response time analysis may conclude that the system is not schedulable even when the resources are severely underutilized. Our preliminary experimental results in Fig. 2 show that when the multiple



(a) two transactions on an ECU network (b) per−job based analysis

Fig. 1. Multiple visit problem.

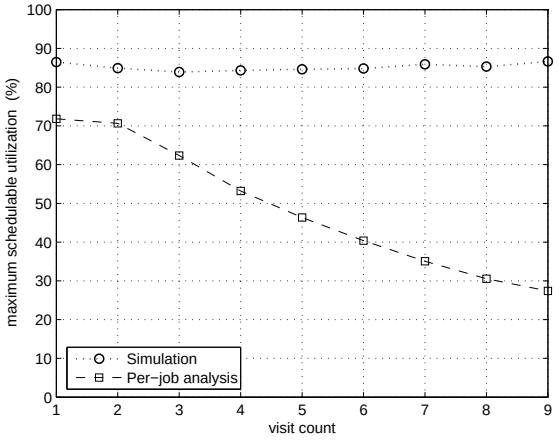


Fig. 2. Maximum schedulable utilization as increasing the transaction length.

visit count on a resource gets larger, the traditional per-job analysis can utilize only up to 30% while the simulation says that the system is still schedulable up to 80% utilization. (The detailed experimental settings will be given later.)

To address this problem, this paper proposes a per-resource based end-to-end response time analysis. The new analysis completely changes the analysis view point from "per-job basis" to "per-resource basis". That is, it computes the total delay at each resource. By adding the total delays at all the resources, we can find a bound of the end-to-end response time. Fig. 3 conceptually compares the per-job based analysis (see the horizontal addition) and the per-resource based analysis (see the vertical addition). The per-resource based analysis does not suffer from "double-counting for multiple visits" and hence gives a much tighter bound on the end-to-end response time, especially when a transaction is complex and long and thus visits the same resource many times.

The rest of this paper is organized as follows: The next section summarizes the related work. Section 3 formally defines the problem and motivates our new analysis. In Section 4, we propose our per-resource based holistic analysis that can find a significantly tighter bound on the end-to-end response time. Section 5 presents the experimental results. Finally, Section 6 concludes this paper.

2 Related Work

The famous classical work [8], [9] presents the worst case response time analysis for multiple tasks on a single processor fixed-priority scheduling system. This analysis is first extended by Tindell for distributed systems with multiple resources [1]. It basically computes the end-to-end response time of a transaction consisting of a sequence of jobs (and messages) over multiple resources by aggregating the per-job response times. Its correctness is revisited by [2].

This end-to-end response time analysis has been improved in many ways. For example, its pessimism due to release jitters is addressed by reducing or eliminating the jitters with a sporadic server, release guards, or best-case response time considerations [4], [10], [11]. The work in [7] explicitly considers precedence relations among jobs in the same transactions and also their priorities to improve the accuracy of analysis. Another group of work [5], [12], [13], [14] explicitly uses time-correlations among jobs, called offsets, in order to less conservatively estimate the preemptions by high priority jobs. This consideration is further improved in [6] by more accurately estimating the offsets relative to arrival/completion times of predecessor tasks instead of single reference point of the external event time.

However, all these works have their basis on Tindell's per-job analysis [1] and hence do not provide a fundamental solution for the aforementioned multiple visit problem.

The delay composition theorem [15] is an innovative idea for analyzing the end-to-end delay of a pipelined distributed system. Unlike the previous work that assumes the worst-case preemption pattern at each stage of a pipelined transaction, the delay composition theorem considers the overlapped executions in different pipeline stages and hence reduces the pessimism of end-to-end delay analysis. More importantly, the theorem provides a way of transforming a pipelined distributed system into a uniprocessor system. Thanks to this transformation, rigorous schedulability analysis techniques developed for uniprocessor systems can be applied to pipelined distributed systems. One limitation, however, is that all the transactions in the system should follow the same path along the same resource stages. This limitation is addressed in [16] by combining transactions following different paths into a Directed Acyclic Graph (DAG) and extending the delay composition theorem to a DAG. Regardless of this extension, it still assumes that each transaction follows an acyclic path without revisiting the same resources. Therefore, the delay composition theorem is not applicable to our target system where transactions visit resources multiple times in arbitrary manners.

There also have been efforts to apply the analysis

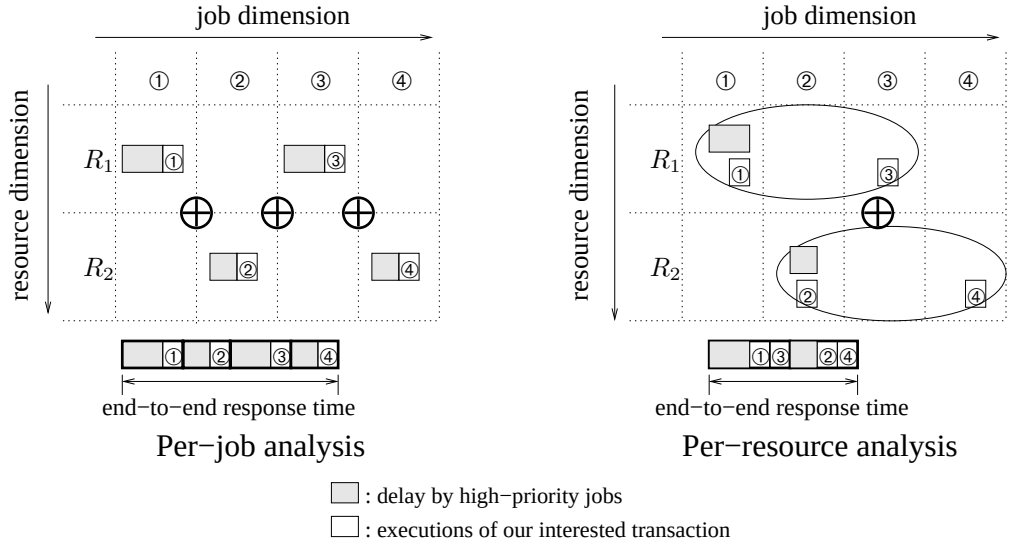


Fig. 3. Conceptual comparison of per-job analysis and per-resource analysis.

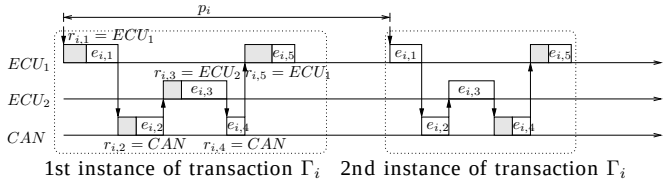


Fig. 4. An example transaction Γ_i.

techniques to automotive applications [17], [18], [19]. The analysis techniques used, however, are again per-job based. Therefore, they are not free from the multiple visit problem either.

In contrast, our analysis proposed in this paper changes the analysis view point to the per-resource basis in order to fundamentally address the pessimism due to the multiple visit problem.

3 Problem Description

In this paper, we consider a system that consists of M resources denoted by $\{R_1, R_2, \cdots, R_M\}$. Some resources are processors and others are communication links to deliver messages among tasks on processors. However, for the simplicity of explanation, we do not differentiate the two resource types assuming that all the resources schedule their jobs (messages in case of a communication link) based on the fixed-priorities preemptive scheduling. Non-preemptive messages on communication links can be simply addressed by counting one message length as a blocking factor [18].

On that system, we assume N periodic transactions denoted by $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$ where Γ_j has a higher priority than Γ_i if $j < i$. Each transaction Γ_i consists of $|\Gamma_i|$ tasks $\{\tau_{i,1}, \tau_{i,2}, \cdots, \tau_{i,|\Gamma_i|}\}$ that are executed on resources $(r_{i,1}, r_{i,2}, \cdots, r_{i,|\Gamma_i|})$ "in sequence" with the worst-case execution times of $(e_{i,1}, e_{i,2}, \cdots, e_{i,|\Gamma_i|})$, respectively. The first task $\tau_{i,1}$ of Γ_i is released periodically with a period of p_i and the subsequent tasks released at the completion times of their immediate predecessor tasks. Thus, a transaction Γ_i can be represented by

$$\Gamma_i = (p_i, \{\tau_{i,1} = (r_{i,1}, e_{i,1}), \tau_{i,2} = (r_{i,2}, e_{i,2}), \cdots, \tau_{i,|\Gamma_i|} = (r_{i,|\Gamma_i|}, e_{i,|\Gamma_i|})\}). \quad (1)$$

One instance of the whole sequence of $\tau_{i,1}, \tau_{i,2}, \cdots, \tau_{i,|\Gamma_i|}$ is called a Γ_i *transaction instance*. Fig. 4 shows two periodic instances of an example transaction Γ_i to visualize the meanings of its notations. Note that each instance of this transaction visits ECU_1 twice, ECU_2 once, and CAN twice. For every instance, the whole sequence of Γ_i's tasks must be completed within a period, that is, the end-to-end deadline is equal to the period p_i. Although we assume that the end-to-end deadline is equal to the period for the simplicity, the proposed analysis still works even when the end-to-end deadline is shorter than the period.

Problem description: For such a given set of N periodic transactions $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$ over M resources $\{R_1, R_2, \cdots, R_M\}$, our problem is to check whether every transaction Γ_i can always meet the end-to-end deadline p_i.

For this problem, we may use the traditional end-to-end response time analysis [1]. Its short overview is given in the following. For each task of a transaction Γ_i, its per-job worst case response time $w_{i,k}$ is calculated with the following response time equation [1]:

$$w_{i,k} = e_{i,k} + \sum_{\forall j<i} \sum_{\forall\{a|r_{j,a}=r_{i,k}\}} \left\lceil \frac{J_{j,a} + w_{i,k}}{p_j} \right\rceil e_{j,a} \quad (2)$$

1. When the deadline is less than or equal to the period, we do not have to consider the delay by the previous instances of the same transaction. Omitting such factor, Equation (2) is a simplified one from the original in [1].

where $J_{j,a}$ is the worst case release jitter of a-th task $\tau_{j,a}$ of a higher priority transaction Γ_j. $J_{j,a}$ is simply given as the worst case response time until the completion of $(a-1)$-th task $\tau_{j,(a-1)}$ of Γ_j. The equation means that the per-job worst case response time of $\tau_{i,k}$ can be calculated by adding (1) its own execution time $e_{i,k}$ and (2) the largest possible delay due to higher priority jobs on the same resource, which is given as $\sum_{\forall j<i} \sum_{\forall\{a|r_{j,a}=r_{i,k}\}} \left\lceil \frac{J_{j,a}+w_{i,k}}{p_j} \right\rceil e_{j,a}$. In the second term, $\left\lceil \frac{J_{j,a}+w_{i,k}}{p_j} \right\rceil$ is the largest number of releases of $\tau_{j,a}$ of Γ_j during the time window $w_{i,k}$ assuming the worst case release pattern where its first release of $\tau_{j,a}$ is delayed the most, i.e., $J_{j,a}$ and succeeding releases are with the maximum rate, i.e., $1/p_j$. Applying the above equation for all the tasks $\tau_{i,k}$ of Γ_i, the worst case end-to-end response time denoted by $e2eRspTime_i$ can be calculated by adding-up all the per-job response times, i.e.,

$$e2eRspTime_i = \sum_{k=1}^{|\Gamma_i|} w_{i,k}. \quad (3)$$

This per-job based analysis can severely overestimate the end-to-end response time when a transaction Γ_i visits the same resource multiple times. In the example of Fig. 4, if we individually apply Equation (2) for $\tau_{i,2}$ and $\tau_{i,4}$ of Γ_i that visit the CAN resource, the worst case delay by higher priority jobs represented by the second term of Equation (2) is counted twice in the final end-to-end response time calculation of Equation (3). The same problem happens for $\tau_{i,1}$ and $\tau_{i,5}$ that visit ECU_1.

Our new analysis aims at addressing this pessimism by changing the analysis view point from *per-job* to *per-resource*.

4 PER-RESOURCE BASED ANALYSIS FOR END-TO-END RESPONSE TIME

For our per-resource based analysis, we introduce a notion of "per-resource total delay". The worst case total delay that one Γ_i instance experiences due to a higher priority transaction Γ_j at resource R_l is denoted by $TD_i^j(R_l)$. Using this notion, the time that one Γ_i instance spends at R_l can be calculated by adding its execution times at R_l and its total delays by all the higher priority transactions at R_l, that is,

$$\left(\sum_{\forall\{(i,k)|r_{i,k}=R_l\}} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_i^j(R_l).$$

Therefore, the end-to-end response time of Γ_i can be calculated by summing up the times spent at all the visiting resources as follows:

$$e2eRspTime_i = \sum_{R_l \in \{R_1,\cdots,R_M\}} \left(\left(\sum_{\forall\{(i,k)|r_{i,k}=R_l\}} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_i^j(R_l) \right). \quad (4)$$

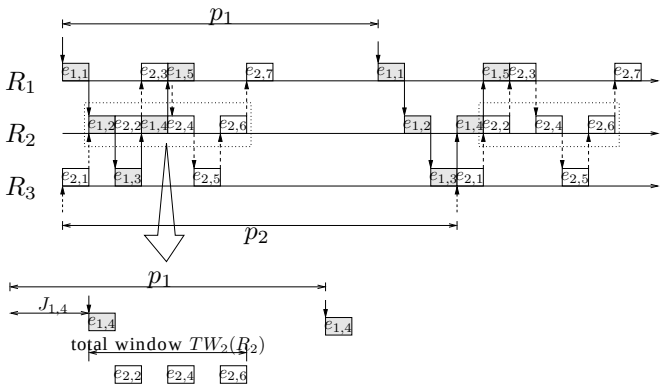


Fig. 5. The concept of total window $TW_i(R_l)$.

Note that for any resource R_l that Γ_i does not visit, both parts of e values and TD values in Equation (4) are zero.

Now, the remaining problem is to find an upper-bound on the per-resource total delay $TD_i^j(R_l)$. In the following, we explain how to find it focusing on a transaction Γ_i assuming that analysis for all the higher priority transactions Γ_j $(j = 1, 2, \cdots, i-1)$ has been completed.

4.1 Total delay bound for a known total window

In order to find an upper-bound on the per-resource total delay $TD_i^j(R_l)$, let us introduce another notion of a "per-resource total window". The per-resource total window denoted by $TW_i(R_l)$ represents the time window during which an instance of a transaction Γ_i has uncompleted jobs to be executed on a resource R_l. In order to explain the concept of the per-resource total window, consider an example in Fig. 5 where two transactions Γ_1 and Γ_2 are concurrently running on three resources, R_1, R_2, and R_3. In this example, the total window of Γ_2 at resource R_2, i.e., $TW_2(R_2)$, is the time span from the release time of its first visit on R_2, i.e., $\tau_{2,2}$, to the completion time of its last visit on R_2, i.e., $\tau_{2,6}$. The total window is depicted as dotted boxes in Fig. 5.

Supposing that we can somehow find an upper bound of the total window $TW_i(R_l)$, the number of instances of a higher priority task $\tau_{j,a}$ that can be released during $TW_i(R_l)$ and hence delay one Γ_i instance at resource R_l (i.e., $r_{j,a} = R_l$) can be upper-bounded by:

$$Z_{j,a}(TW_i(R_l)) = \left\lceil \frac{J_{j,a} + TW_i(R_l)}{p_j} \right\rceil \quad (5)$$

assuming the worst case release pattern of $\tau_{j,a}$, i.e., the first release is delayed the most by the amount of jitter $J_{j,a}$ and subsequent releases are most packed with the maximum release rate of $1/p_j$. Later, we will explain how to calculate $J_{j,a}$ in Section 4.3. In the example of Fig. 5, considering p_1, the maximum number of instances of $\tau_{1,4}$ during $TW_2(R_2)$ is one. In contrast, the per-job analysis counts one instance for each job of three visits on R_2, i.e.,

$\tau_{2,2}$, $\tau_{2,4}$, and $\tau_{2,6}$, and thus totally three instances are pessimistically counted in the final end-to-end response time of Γ_2.

Now, for the known duration of total window, i.e., $TW_i(R_l)$, we are clear in that no more than $Z_{j,a}(TW_i(R_l))$ instances of $\tau_{j,a}$ can delay one Γ_i instance at resource R_l. However, if we count all of $Z_{j,a}(TW_i(R_l))$ instances as contributions to the total delay $TD_i^j(R_l)$, it would be too pessimistic, especially when the period of a high priority transaction Γ_j is much shorter than the total window $TW_i(R_l)$. As an example, Fig. 6 shows two cases: (1) Case I where the high priority period p_j is longer than the total window $TW_i(R_l)$ and (2) Case II where p_j is shorter than $TW_i(R_l)$. In Case I, it is okay to count $Z_{j,a}(TW_i(R_l)) = 1$ in the total delay estimation. In Case II, however, it is pessimistic to count all the 4 instances of $\tau_{j,a}$ as contributions to the total delay, since Γ_i may not be "busy"—having jobs released but not completed, in the whole duration of $TW_i(R_l)$.

In both Cases I and II, in order to more tightly but still sufficiently count the number of instances of $\tau_{j,a}$ instances that should be included in the total delay, we compute the worst case per-job busy interval one-by-one by using $Z_{j,a}(TW_i(R_l))$ as the limit of delay as follows:

$$w_{i,k} = e_{i,k} + \sum_{\forall j<i} \sum_{\forall\{a|r_{j,a}=r_{i,k}=R_l\}} \left(\min\left(\left\lceil \frac{J_{j,a}+w_{i,k}}{p_j} \right\rceil, Z_{j,a}(TW_i(R_l)) \right) e_{j,a} \right). \quad (6)$$

In Case I of Fig. 6, if we apply the above equation to the first job with $Z_{j,a}(TW_i(R_l)) = 1$, the one $\tau_{j,a}$ instance is included in the first busy interval and there is no more $\tau_{j,a}$ instance in the duration of the total window $TW_i(R_l)$. Thus, when we apply the above equation to the second and third jobs of Γ_i, $Z_{j,a}(TW_i(R_l)) = 0$ is used. Thus, no more instances of $\tau_{j,a}$ are counted for the second and third job delays. Overall, only one $\tau_{j,a}$ instance is counted for all the three jobs of Γ_i. Although the Case I of Fig. 6 shows a specific scenario where the first visit of Γ_i is delayed by the single instance of $\tau_{j,a}$, it does not matter which visit is actually delayed. The thing that matters is that the total delay by $\tau_{j,a}$ that a single Γ_i instance experiences at resource R_l is upper-bounded by one $e_{j,a}$. This is the case where we can find a tighter estimation of the total delay than the per-job analysis [1] that counts one $e_{j,a}$ for each of the three jobs.

In Case II of Fig. 6, when we apply the above equation to the first job, the given value of 4 is used as $Z_{j,a}(TW_i(R_l))$. When calculating the first job's busy interval, we assume the worst case release pattern of the four $\tau_{j,a}$-instances as shown in the middle of the Case II figure. Hence, we include $J_{j,a}$ in the ceiling function of Equation (6). This way, we can maximally include $e_{j,a}$s in the first job's busy interval, regardless of various release scenarios. Once the above recursive equation converges, we can notice how many $e_{j,a}$s are included in the busy interval of the first job of Γ_i on R_l. Suppose that the number is one as shown in Case II of the figure. Then, we apply the above equation to the second job of Γ_i on R_l with left-over instances of $e_{j,a}$, that is, $Z_{j,a}(TW_i(R_l)) = 3$. For the second job, we again assume the worst case release pattern of the three $\tau_{j,a}$-instances as shown in the bottom of the Case II figure using $J_{j,a}$ again in Equation (6). When the recursive equation converges, we can notice the maximal number of $e_{j,a}$s that are included in the second job's busy interval. Suppose that the number is one as shown in Case II of the figure. Then, the left-over two instances of $\tau_{j,a}$ have no way to be included in the total delay $TD_i^j(R_l)$, because all the R_l visiting jobs of Γ_i have already included maximal delay effects by $\tau_{j,a}$ assuming the worst case, just like the per-job analysis [1]. Therefore, we can sufficiently count only two as contributions to the total delay $TD_i^j(R_l)$ as shown in Case II of the figure. This is the case where we find the same estimation of the total delay as the per-job analysis [1].

This way, in both Cases I and II, the delay contributions counted in our analysis is always smaller than or equal to that of the per-job analysis by Tindell [1].

Applying Equation (6) for all the tasks $\tau_{i,k}$ of Γ_i that visit R_l, we can obtain the worst case numbers of instances of $e_{j,a}$ that can contribute to the total delay $TD_i^j(R_l)$ within the given duration of the total window $TW_i(R_l)$. Denoting such a number of instances of $e_{j,a}$ by $C_i^{j,a}(TW_i(R_l))$, the total delay $TD_i^j(R_l)$ of Γ_i caused by Γ_j at resource R_l for the given total window $TW_i(R_l)$ can be computed as follows:

$$TD_i^j(R_l) = \sum_{\forall\{a|r_{j,a}=R_l\}} \left(C_i^{j,a}(TW_i(R_l)) \times e_{j,a} \right). \quad (7)$$

Lemma 1. *$TD_i^j(R_l)$ computed by Equation (7) is an upper-bound of the total delay that an instance of Γ_i experiences by tasks of Γ_j at resource R_l during the given total window $TW_i(R_l)$.*

Proof: If we omit $Z_{j,a}(TW_i(R_l))$ from Equation (6), the resulting count $C_i^{j,a}(TW_i(R_l))$ used in Equation (7) is exactly the same as the Tindell's count [1]. Note that, for each task $\tau_{i,k}$ of Γ_i that visits R_l, Tindell's count assumes the worst case release jitter of $\tau_{j,a}$ as we can see in the ceiling part of Equation (6). Thus, Tindell's count is an upper bound of the number of instances of $\tau_{j,a}$ that can delay all the tasks $\tau_{i,k}$ of Γ_i that visit R_l, as proven in [1]. Only if Tindell's count is larger than $Z_{j,a}(TW_i(R_l))$ (as in Case I of Fig. 6), Equation (6) gives $Z_{j,a}(TW_i(R_l))$ as the value of $C_i^{j,a}(TW_i(R_l))$. Thus, it is sufficient to prove that $Z_{j,a}(TW_i(R_l))$ is also an upper-bound of the number of $\tau_{j,a}$ instances that can delay all the tasks $\tau_{i,k}$ of one Γ_i instance at R_l. Supposing that $TW_i(R_l)$ is an upper bound of the total window of Γ_i at R_l, we can pessimistically consider the whole duration of $TW_i(R_l)$ as the worst case busy period of Γ_i at R_l. Therefore, $Z_{j,a}(TW_i(R_l))$ computed by Equation (5) safely upper bounds the number of $\tau_{j,a}$ instances that can delay the execution of Γ_i during its busy period

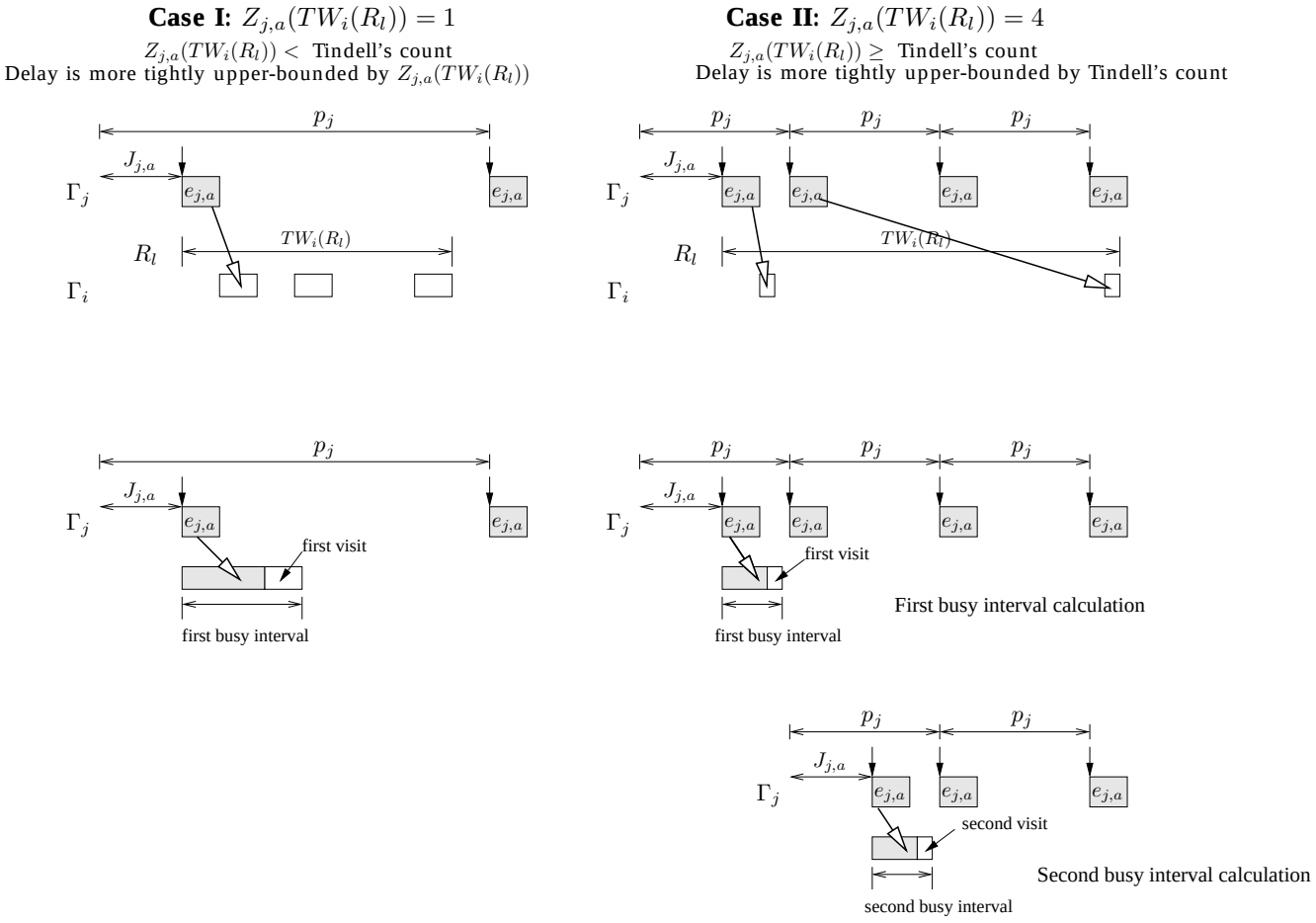


Fig. 6. Contributions to the total delay $TD_i^j(R_l)$.

$TW_i(R_l)$, as proven in [1]. With the above two arguments, $C_i^{j,a}(TW_i(R_l))$ counted using Equation (6) is an upper bound of $\tau_{j,a}$ instances that can delay one Γ_i instance at R_l during the duration of $TW_i(R_l)$. Since it holds for every task $\tau_{j,a}$ of Γ_j, $TD_i^j(R_l)$ computed by Equation (7) is an upper-bound of the total delay that one Γ_i instance experiences by tasks of Γ_j at resource R_l during the given total window $TW_i(R_l)$. □

As a side note, if we additionally consider inter-task offsets of a higher priority transaction as in [5], [6], [12], counting the contributions to the total delay can be more accurate. This improvement will be made in our future work.

4.2 Iterative calculation of total delay and total window

The total delay $TD_i^j(R)$ and the total window $TW_i(R)$, in fact, are inter-dependent. To find an upper bound of $TD_i^j(R)$ addressing the inter-dependency, we use an iterative convergence approach, which is similar to the iterative solving of traditional recursive response time equation [8], [9].

Initially, we set $TD_i^j(R) = 0$ for all the high priority transactions $\Gamma_j \in \{\Gamma_1, \cdots, \Gamma_{i-1}\}$ and for all the resources $R \in \{R_1, \cdots, R_M\}$.

Once the total delay values $TD_i^j(R)$ for all j and R are given (initial values of them are zero), we can compute an upper-bound of the total window $TW_i(R)$ for all R. To explain this, let us denote the Γ_i's tasks visiting R as

$$(\tau_{i,v_1}, \tau_{i,v_2}, \cdots, \tau_{i,v_m})$$

where $r_{i,v_1} = r_{i,v_2} = \cdots = r_{i,v_m} = R$ and $v_1 < v_2 < \cdots < v_m$. Using this notation, an upper-bound of Γ_i's total window at R, i.e., $TW_i(R)$, can be computed as follows:

$$TW_i(R) = \sum_{v_1 \leq k \leq v_m} e_{i,k} + \sum_{\forall R_l \in \{R_1, \cdots, R_M\}} \sum_{j=1}^{i-1} TD_i^j(R_l) X_{v_1}^{v_m}(R_l) \quad (8)$$

where $X_{v_1}^{v_m}(R_l) = \begin{cases} 1 & \text{if there is a visit on } R_l \text{ in the subsequence from } \tau_{i,v_1} \text{ to } \tau_{i,v_m} \\ 0 & \text{otherwise} \end{cases}$

This equation can be best explained with Fig. 7. Suppose that total delays so far at all resources R_1, R_2, R_3, and R_4 are given as the shaded boxes in the

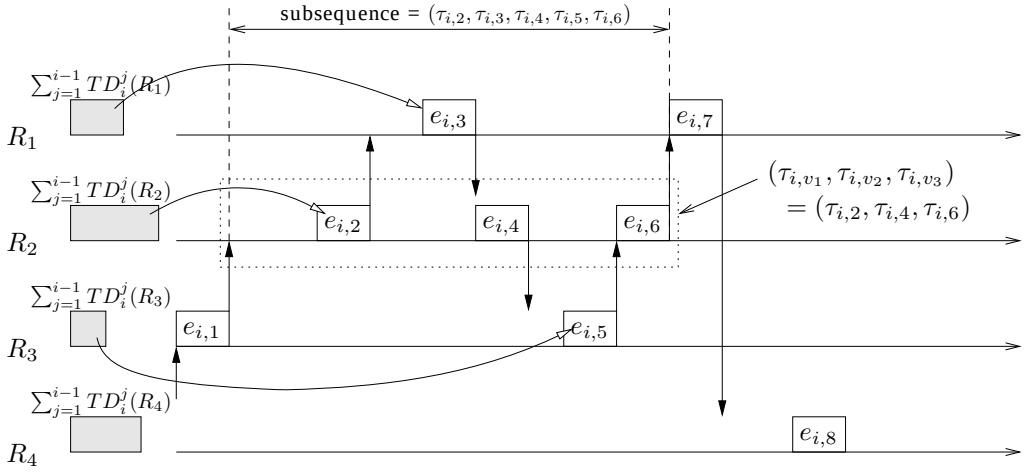


Fig. 7. Total window estimation $TW_i(R_2)$ from total delays.

figure. Then, the total window $TW_i(R_2)$ at R_2 is given by adding (1) the execution times of the subsequence $(\tau_{i,2}, \tau_{i,3}, \tau_{i,4}, \tau_{i,5}, \tau_{i,6})$ from the first visit on R_2, i.e., $\tau_{i,2}$, to the last visit on R_2, i.e., $\tau_{i,6}$, and (2) the delays that Γ_i experiences while executing the subsequence. The former can simply be represented by the first term of Equation (8) which adds all the execution times of the subsequence, i.e., $\sum_{2 \le k \le 6} e_{i,k} = e_{i,2}+e_{i,3}+e_{i,4}+e_{i,5}+e_{i,6}$. The latter can be calculated by conservatively assuming that all the portions of the total delays happen within the subsequence as depicted in the figure. This is represented by the second term of Equation (8). Note that there is no visit on R_4 within the subsequence. Thus, $X_2^6(R_4) = 0$ by definition of X. This prevents the total delay at R_4 from being added in the second term of Equation (8).

Lemma 2. *Supposing that $TD_i^j(R_l)$ is an upper-bound of the total delay of Γ_i by Γ_j at R_l for every j and R_l, $TW_i(R)$ computed by Equation (8) is an upper-bound of the total window of Γ_i at R.*

Proof: The total window of Γ_i at R starts from the release time of τ_{i,v_1} and ends at the completion time of τ_{i,v_m}. In between these two time points, Γ_i may visit other resources and revisit R as shown in Fig. 7. Thus, the total window is given as the sum of (1) execution times of all the tasks of Γ_i from τ_{i,v_1} to τ_{i,v_m}—$e_{i,2} + e_{i,3} + e_{i,4} + e_{i,5} + e_{i,6}$ in Fig. 7 and (2) delays they experience at their visiting resources by higher priority tasks—see Fig. 7. The former can be exactly counted by the first term of Equation (8). Thus, the remaining problem is to show that the second term of Equation (8) is an upper-bound of the delay part. Since $TD_i^j(R_l)$ is an upper-bound of the total delay by Γ_j that the whole task sequence of one Γ_i instance experiences at R_l, it also upper-bounds the delay that tasks within the subsequence experience at R_l by Γ_j. By summing up $TD_i^j(R_l)$s for all visiting resources R_l and for all higher priority transactions Γ_j, the second term of Equation (8) upper-bounds the delay part. Therefore, $TW_i(R)$ computed by Equation (8) is an upper-bound of the total window of Γ_i at R. □

Once we have the total windows for all the resources, i.e., $TW_i(R_1), TW_i(R_2), \cdots, TW_i(R_M)$, we can compute new upper-bounds of total delays, i.e., $TD_i^j(R_1), TD_i^j(R_2), \cdots, TD_i^j(R_M)$ for all $j \in \{1, \cdots, i-1\}$ with Equations (6) and (7). If any of these new total delays is larger than its previous value, we continue the iterations. When all the total delay values no longer increase, that is, "convergence", we terminate the iteration. With the converged values of the total delays $TD_i^j(R_l)$, we can finally compute an upper-bound of the end-to-end response time of Γ_i, i.e., $e2eRspTime_i$, based on Equation (4).

Theorem 1. *$e2eRspTime_i$ computed by Equation (4) is an upper-bound of the end-to-end response time of one Γ_i instance.*

Proof: $TW_i(R_l)$ for all R_l and $TD_i^j(R_l)$ for all j and R_l are monotonically non-decreasing. Due to this fact, and by Lemmas 1 and 2, the converged value of $TD_i^j(R_l)$ for every j and R_l is an upper-bound of the total delay that one Γ_i instance experiences by tasks of Γ_j at resource R_l. Therefore, $\left(\sum_{\forall \{(i,k) | r_{i,k} = R_l\}} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_i^j(R_l)$ is an upper-bound of the total time that one Γ_i instance spends at resource R_l. Consequently, $\sum_{R_l \in \{R_1, \cdots, R_M\}} \left(\left(\sum_{\forall \{(i,k) | r_{i,k} = R_l\}} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_i^j(R_l) \right)$ in Equation (4) gives an upper-bound of the total time that one Γ_i instance spends at all its visiting resources. (Note that for a non-visiting resource R_l, both parts of e values and TD values in Equation (4) are zero.) Therefore, the theorem holds. □

TABLE 1
Experimental Parameters

parameter	value
number of resources M	10 (1 CAN and 9 ECUs)
number of transactions N	5
transaction period p_i	uniform from $[p_{min}, p_{max}]$
transaction length $\|\Gamma_i\|$	L
task sequence $\{\tau_{i,1}, \cdots, \tau_{i,L}\}$	random ECU and CAN alternating
task execution time $e_{i,k}$	uniform from [1 ms, 5 ms]

4.3 Jitter calculation for analysis of low priority transactions

For analyzing a low priority transaction, we have to compute the worst case release jitters for all the tasks $\{\tau_{i,1}, \tau_{i,2}, \cdots, \tau_{i,|\Gamma_i|}\}$ of Γ_i. Since a task $\tau_{i,k}$ is released by the completion of its immediate predecessor task $\tau_{i,k-1}$, its worst case release jitter $J_{i,k}$ can be given as the worst case response time until the completion of $\tau_{i,k-1}$. It can simply be calculated by applying the above per-resource based end-to-end response time analysis to the subsequence $(\tau_{i,1}, \cdots, \tau_{i,k-1})$.

5 EXPERIMENTS

This section validates our proposed analysis in terms of the analysis accuracy. For this, we consider an automotive-style resource model with 10 resources, i.e., $\{R_1, R_2, \cdots, R_{10}\}$, one of which is a CAN bus and other 9 resources are ECUs communicating each other through the CAN bus. On top of this resource model, we assume five periodic transactions $\{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5\}$. The set of five periodic transactions is randomly generated as follows: The period of each transaction is randomly selected from the range of $[p_{min}, p_{max}]$ following the uniform distribution. The priority of a transaction is assigned according to the period, that is, a transaction with a shorter period is assigned with a higher priority. Every transaction has the same length L—the number of tasks $|\Gamma_i|$ is L for all $\Gamma_i \in \{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5\}$. The first task of a transaction is mapped to an ECU randomly selected out of 9 ECUs. The second task is mapped to the CAN modeling the message transmission to the third task. Then, the third task is again mapped to a randomly selected ECU and so on, until we make a sequence of L tasks. By increasing the transaction length L, we can control the visit count on CAN. The execution time of a task mapped on a resource is randomly selected from the range of [1 ms, 5 ms]. Table 1 summarizes these experimental parameters. The results in the following are the averages for such 300 generated random sets.

With these parameters, we compare four analysis methods:

- Tindell's per-job analysis [1] denoted by $Tindell$,
- Palencia's and Turja's enhanced per-job analysis considering inter-task offsets [5], [12] [2] denoted by $WCDO$ meaning "Worst Case Dynamic Offset",
- Henia's further improved analysis [6] denoted by $Henia$, and
- Our proposed per-resource analysis denoted by $OurPerResource$.

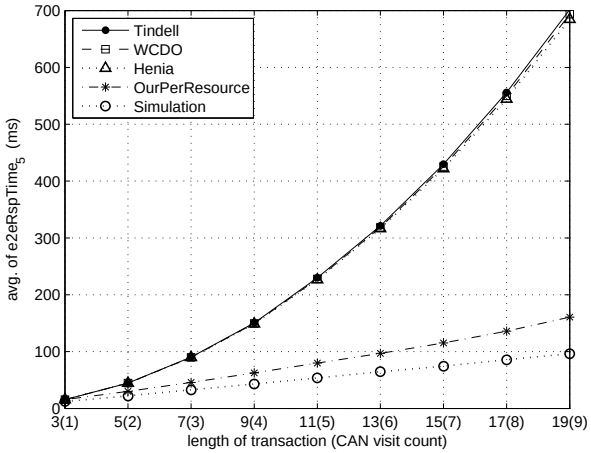


Fig. 8. End-to-end response time as increasing the transaction length.

We also present simulations results. Although the simulation does not give upper-bounds of end-to-end response times, it can give typical end-to-end response times. We use the simulation results to see the overestimation by the analysis methods.

Fig. 8 compares the end-to-end response time $e2eRspTime_5$ of the lowest priority transaction Γ_5, as increasing the transaction length L and hence the visit count on CAN. The x-axis shows L together with the CAN visit count in the parenthesis. In this experiment, the period selection range $[p_{min}, p_{max}]$ is fixed as [100 ms, 1000 ms]. When the transaction length is short, all of the above four analysis methods give a pretty tight bound on the end-to-end response time, which is close to the one by simulation. As increasing the transaction length, the end-to-end response time increases since the average workload on all the resources increases. However, the increase rates are quite different. All of the per-job based analysis methods, i.e., $Tindell$, $WCDO$, and $Henia$, show a sharp increase of the end-to-end response time due to many double-counts for increasing multiple visits. On the other hand, our per-resource analysis suffer less from the double-counting problem and thus shows a much less increase of the end-to-end response time. Consequently, when the transaction length is 19, the end-to-end response time by our analysis is over four times shorter than those by per-job analysis methods. Moreover, our result is quite close to the simulation result.

2. We use Turja's method [12] since it is most up-to-date.

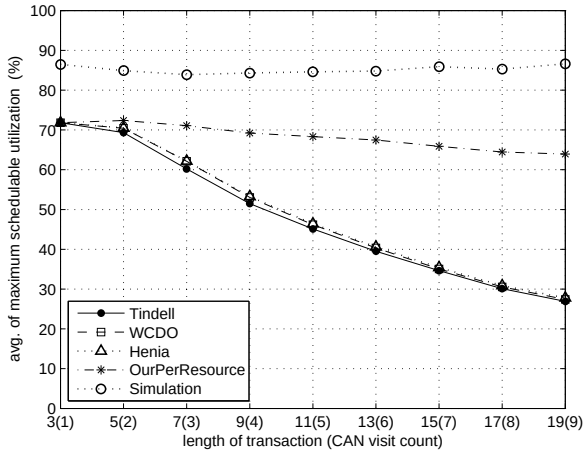


Fig. 9. Maximum schedulable utilization as increasing the transaction length.

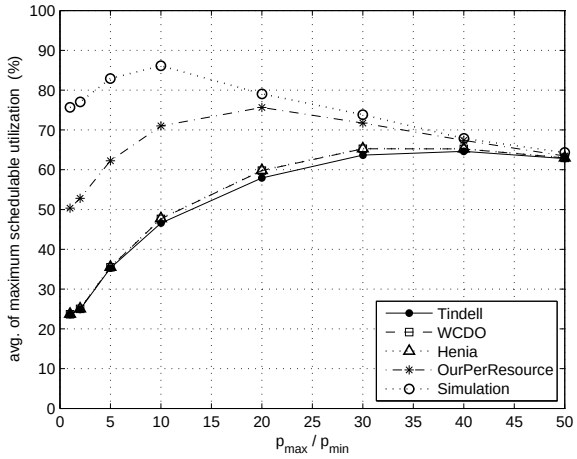


Fig. 10. Maximum schedulable utilization as increasing the period ratio.

In order to see how much we can utilize the resources under the schedulability constraint, we perform another experiment as scaling-up all the execution times until the system becomes unschedulable. At the saturation point, we observe the utilization of the resource with the largest utilization, which we call a *maximum schedulable utilization*. Fig. 9 compares the maximum schedulable utilization as increasing the transaction length L with the period selection range of [100 ms, 1000 ms]. As expected by Fig. 8, when the transaction length is long, say 19, the maximum schedulable utilizations by the per-job analysis methods are below 30%. On the other hand, our analysis can achieve the maximum schedulable utilization above 60%. This implies that, with our analysis, industries can better utilize their given resources by accommodating more transactions for advanced features. The other way around is also true—the same set of transactions can be implemented with lower-speed resources, which can save the unit cost of production.

Another important factor that affects the analysis accuracy is the period ratio of high and low priority transactions. In order to study this factor, Fig. 10 compares the maximum schedulable utilization as varying the period selection range $[p_{min}, p_{max}]$ from [100 ms, 100 ms] to [100 ms, 5000 ms] while fixing the transaction length $L = 10$. When the period ratio p_{max}/p_{min} is not that large, our per-resource based analysis can significantly reduce the double counts made by per-job analysis since Case I of Fig. 6 is common. Thus, our per-resource analysis can achieve a much higher maximum schedulable utilization than the per-job analysis methods, i.e., $Tindell$, $WCDO$ and $Henia$. However, as the period ratio p_{max}/p_{min} becomes large, Case II of Fig. 6 becomes common and thus the gap decreases. When the period ratio p_{max}/p_{min} is extremely large as 50, our analysis eventually degenerates into $Tindell$. However, in practical applications such as automotive systems, many transactions have comparable periods with reasonably small period ratios. Thus, our analysis can have significant improvements in many practical settings.

One interesting observation from the above experiments is that there is no significant improvement by $WCDO$ and $Henia$ over $Tindell$. This is because the improvement is possible only when a low priority job can be fit into the inter-task time gaps, called offsets, of a high priority transaction. Such cases did not commonly happen in our previous experimental setting. In order to give a favor to $WCDO$ and $Henia$, in the next experiment, we pick execution times of the lowest priority transaction Γ_5 from very short values in [0.01 ms, 0.05 ms]. In addition, for other transactions to have large inter-task offsets on CAN, their execution times on ECUs are picked from large values in [10 ms, 50 ms] but those on CAN are picked from medium values in [1 ms, 5 ms]. With this special setting, Fig. 11 compares the end-to-end response time $e2eRspTime_5$ of the lowest priority transaction Γ_5 as increasing the transaction length of the

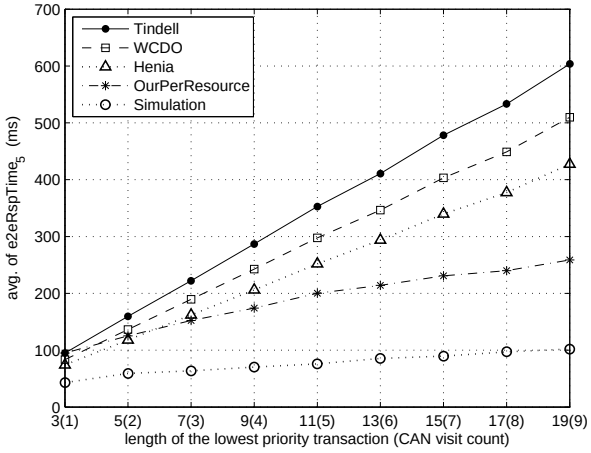


Fig. 11. Experiment giving a favor to WCDO.

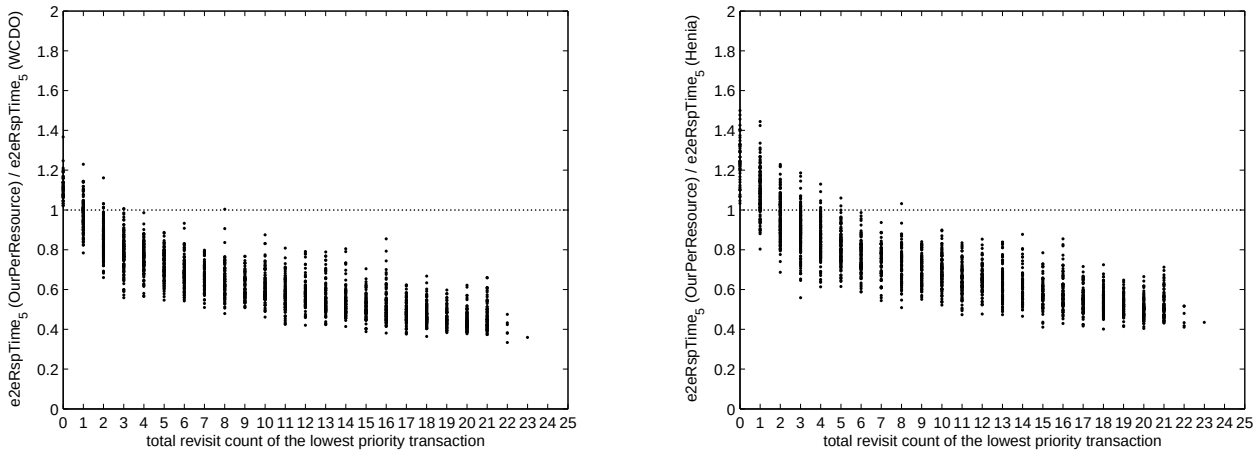


Fig. 12. The ratio of end-to-end responses time by (a) $OurPerResource$ over $WCDO$. (b) $OurPerResource$ over $Henia$.

lowest priority transaction while fixing other transaction lengths as 10. When the length of the lowest priority transaction is 3, it visits an ECU, CAN, and another ECU, in sequence. Thus, it does not have any multiple visits. In this case, our per-resource analysis degenerates into $Tindell$ since we do not take advantage of inter-task offsets in the total delay estimation of Equation (6). On the other hand, $WCDO$ and $Henia$ explicitly consider the inter-task offsets and hence gives a slightly better estimation than ours. However, as soon as the lowest priority transaction length becomes 5 visiting CAN twice, the double count reduction by our per-resource analysis catches up the benefit of the offset consideration by $WCDO$ and $Henia$. After that, our per-resource analysis gives significantly better results than $WCDO$ and $Henia$.

In order to further compare $OurPerResource$ with $WCDO$ and $Henia$, we now consider a new resource model consisting of 10 resources of the same type (e.g., processors). Transactions can visit the resources in any sequence instead of alternating ECUs and CAN. We also generate sets of 5 transactions in a purely random way: (1) each transaction length is randomly picked from [5, 30], (2) each transaction period is randomly picked from [100 ms, 1000 ms], (3) each task of a transaction is mapped to a randomly picked resource out of 10 resources, and (4) each task's execution time is randomly picked from [1 ms, 5 ms]. The priorities of the transactions are determined according to their random periods. Fig. 12 shows the results for 2000 random sets. Each dot in Fig. 12(a) represents a random set showing the ratio of $e2eRspTime_5$ by $OurPerResource$ over that of $WCDO$. On the other hand, each dot in Fig. 12(b) shows the same ratio between $OurPerResource$ and $Henia$. The x-axis is the total count that the lowest priority transaction visits the same resources more than once. When the lowest priority transaction length is small without any multiple visit, $WCDO$ and $Henia$ are always better than $OurPerResource$. However, as we increase the length of the lowest priority transaction making many revisits, our analysis is significantly better than $WCDO$ and $Henia$ in most cases. In addition, there is a potential that our per-resource analysis can be further improved by taking the idea of $WCDO$ and $Henia$ in the total delay estimation of Equation (6) since the total delay estimation is an orthogonal issue to the view point change from per-job to per-resource.

6 Conclusion

In this paper, we propose a fundamental change from the per-job based analysis to the per-resource based analysis to find a tighter bound on the end-to-end response time of a real-time transaction over multiple resources. Instead of aggregating the per-job response times, our proposed analysis aggregates the per-resource total delays. An iterative convergence method is proposed to find the per-resource total delays at all the resources and in turn the end-to-end response time.

The proposed analysis handles the pessimism caused by the multiple visit problem. Therefore, it can significantly improve the analysis accuracy of the end-to-end response times, especially when there are complex long transactions and thus they visit the same resources many times. Our extensive analysis shows that, when the multiple visit count is large, the proposed per-resource analysis can reduce up to 77% of the end-to-end response time estimation by existing per-job analysis methods. This improvement of estimation makes the mathematical timing analysis to be practically applicable to emerging distributed real-time application domains such as autonomously driving vehicles and collaborating autonomous robots.

In the future, we plan to extend our per-resource analysis by addressing deadlines longer than periods and also by further improving the accuracy of total delay estimations considering inter-task offsets as in WCDO [5], [6], [12]. We also plan to investigate the possibility of combining the idea of capturing pipelining effects as in [15], [16] with our per-resource analysis, in order to fundamentally address the pessimism of our jitter-based total delay estimation. Another direction of our future research is to apply the proposed analysis technique to the actual real-time transactions in automotive systems.

ACKNOWLEDGMENTS

This work was supported in part by the IT R&D program of MKE/IITA [2008-F-045-02]. The corresponding author is Chang-Gun Lee.

REFERENCES

[1] K. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," *Microprocessing and Microprogramming - Euromicro Journal*, vol. 40, no. 2-3, pp. 117–134, April 1994.

[2] J. C. Palencia, J. J. G. García, and M. G. Harbour, "On the Schedulability Analysis for Distributed Hard Real-Time Systems," in *Proc. of the 9th Euromicro Workshop on Real-Time Systems*, 1997, pp. 136–143.

[3] S. Bradley, W. Henderson, and D. Kendall, "Reducing conservatism in response time analysis of distributedsystems," in *IEE Colloquium on Applicable Modelling, Verification and Analysis Techniques for Real-Time Systems*, Jan. 1999, pp. 7/1–7/4.

[4] J. J. G. García and M. G. Harbour, "Increasing schedulability in distributed hard real-time systems," in *Proc. of the 7th Euromicro Workshop on Real-Time Systems*, 1995, pp. 99–106.

[5] J. C. Palencia and M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," in *Proc. of the 19th IEEE Real-Time Systems Symp.*, Dec. 1998, pp. 26–37.

[6] R. Henia and R. Ernst, "Improved offset-analysis using multiple timing-references," in *Proc. of the conference on Design, Automation and Test in Europe*, March 2006, pp. 450–455.

[7] J. C. Palencia and M. G. Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *Proc. IEEE 20th Real-Time Systems Symp.*, Dec. 1999, pp. 328–339.

[8] K. Tindell, A. Burns, and A. Wellings, "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks," *J. Real-Time Systems*, vol. 6, no. 2, pp. 133–151, March 1994.

[9] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *BCS Computer J.*, vol. 29, no. 5, pp. 390–395, Oct. 1986.

[10] J. Sun and J. Liu, "Bounding the end-to-end response times of tasks in a distributed real-time system using the direct synchronization protocol," Department of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep. UIUCDCS-R-96-1949, June 1996.

[11] J. C. Palencia, J. J. G. García, and M. G. Harbour, "Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems," in *Proc. of the 10th Euromicro Workshop on Real-Time Systems*, 1998, pp. 35–44.

[12] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real-Time Systems*, vol. 40, no. 1, pp. 77–116, 2008.

[13] O. Redell, "Analysis of three-shaped transactions in distributed real time systems," in *Proc. of the 16th Euromicro Conference on Real-Time Systems*, June 2004.

[14] O. Redell and M. Torngren, "Calculating Exact Worst Case Response Times for Static Priority Scheduled Tasks with Offsets and Jitter," in *Proc. of the 8th Real-Time and Embedded Technology and Applications Symposium*, Sept 2002, pp. 164–172.

[15] P. Jayachandran and T. Abdelzaher, "A Delay Composition Theorem for Real-Time Pipelines," in *Proc. of the 19th Euromicro Conference on Real-Time Systems*, July 2007.

[16] P. Jayachandran and T. Abdelzaher, "Transforming Distributed Acyclic Systems into Equivalent Uniprocessors Under Preemptive and Non-Preemptive Scheduling," in *Proc. of the 20th Euromicro Conference on Real-Time Systems*, July 2008.

[17] L. Pinho and F. Vasques, "Timing Analysis of Reliable Real-Time Communication in CAN Networks," in *Proc. of the 13th Euromicro Workshop on Real-Time Systems*, 2001.

[18] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, April 2007.

[19] K. Tindell, A. Burns, and A. Wellings, "Calculating Controller Area Network(CAN) message response times," *Control Engineering Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

Man-Ki Yoon received the BS degree in Computer Science and Engineering from Seoul National University, Korea, in 2009. He is currently working toward the MS degree in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His current research interests include real-time/embedded systems, cyber-physical systems, real-time scheduling, assumption management for safety-critical systems. He is a student member of IEEE.

Chang-Gun Lee Chang-Gun Lee received the BS, MS and Ph.D. degrees in Computer Engineering from Seoul National University, Korea, in 1991, 1993 and 1998, respectively. He is currently an Associate Professor in the School of Computer Science and Engineering, Seoul National University, Korea. Previously, he was an Assistant Professor in the Department of Electrical and Computer Engineering, The Ohio State University, Columbus from 2002 to 2006, a Research Scientist in the Department of Computer Science, University of Illinois at Urbana-Champaign from 2000 to 2002, and a Research Engineer in the Advanced Telecomm. Research Lab., LG Information and Communications, Ltd. from 1998 to 2000. His current research interests include real-time systems, complex embedded systems, cyber-physical systems, ubiquitous systems, QoS management, wireless ad-hoc networks, and flash memory systems. Dr. Lee is a member of the IEEE and the IEEE Computer Society.

Junghee Han Junghee Han is currently an assistant professor at Korea Aerospace University. She was a senior engineer at Samsung Electronics and a research specialist of Biomedical Informatics at the Ohio State University. She received her Ph.D. from the University of Michigan and her M.S and B.S from Seoul National University. Her research interests include grid computing, network security, and wireless sensor networks.